

初识 ElasticSearch



倒排索引

倒排索引：将各个文档中的内容，进行分词，形成词条。然后记录词条和数据的唯一标识（id）的对应关系，形成的产物。

Key (term)	Value
床	《静夜思》
前	《静夜思》
床前	《静夜思》
明月	《静夜思》，《水调歌头》
月	《静夜思》，《水调歌头》
光	《静夜思》
月光	《静夜思》
几时	《水调歌头》
有	《水调歌头》

PowerPoint 幻灯片放映 - [ElasticSearch.ppt] [兼容模式] - Microsoft PowerPoint

初识 ElasticSearch

ElasticSearch数据的存储和搜索原理

The diagram illustrates the storage and search mechanism of Elasticsearch. It shows three documents (华为主机, 三星移动4G手机, 小米移动4G手机) being indexed into an index (索引库). The index contains terms and their document IDs.

term	Value
华为	1,3
电信	1
3G	1
手	1,2,3
手机	1,2,3
三星	2
移动	2,3
4G	2,3

ElasticSearch概念

- ElasticSearch是一个基于Lucene的搜索服务器
- 是一个分布式、高扩展、高实时的搜索与数据分析引擎
- 基于RESTful web接口
- Elasticsearch是用Java语言开发的，并作为Apache许可条款下的开放源码发布，是一种流行的企业级搜索引擎
- 官网：<https://www.elastic.co/>

应用场景

- 搜索：海量数据的查询
- 日志数据分析
- 实时数据分析

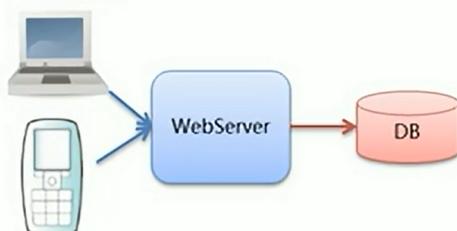


ElasticSearch概念

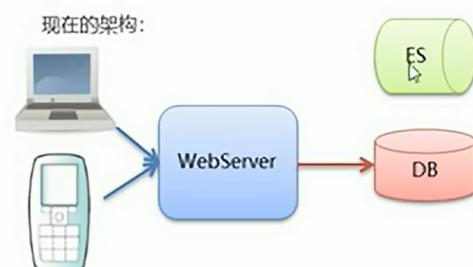
- MySQL有事务性,而ElasticSearch没有事务性,所以你删了的数据是无法恢复的。
- ElasticSearch没有物理外键这个特性,如果你的数据强一致性要求比较高,还是建议慎用

ElasticSearch和MySQL分工不同, MySQL负责存储数据, ElasticSearch负责搜索数据。

以前的架构:



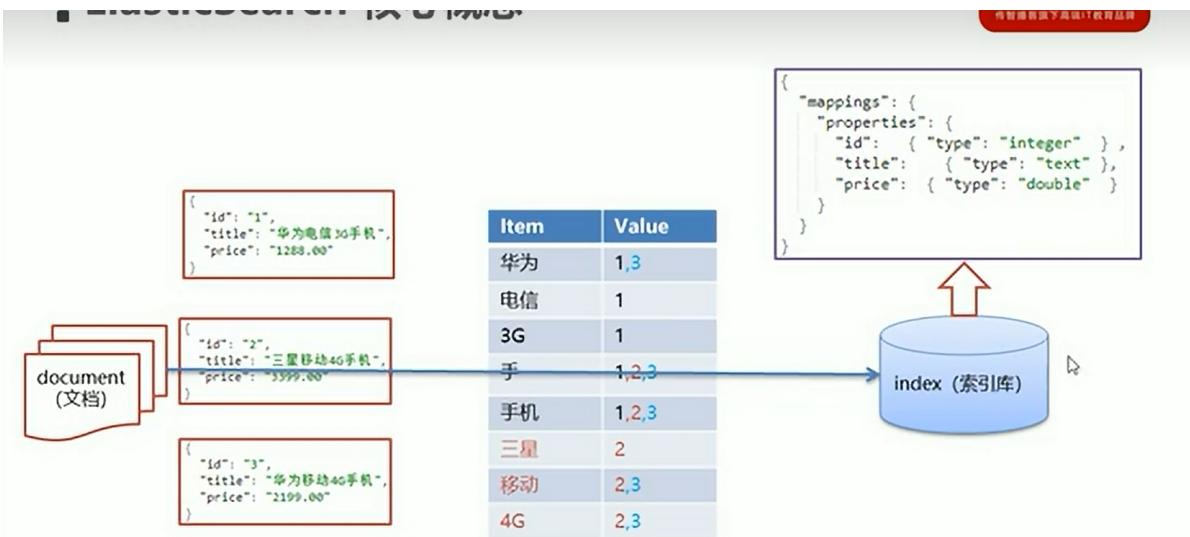
现在的架构:



ElasticSearch核心概念

- 索引 (index)
- 类型 (type)
- 映射 (mapping) 
- 文档 (document)
- 倒排索引





● 索引 (index)

ElasticSearch存储数据的地方，可以理解成关系型数据库中的数据库概念。

● 映射 (mapping)

mapping定义了每个字段的类型、字段所使用的分词器等。相当于关系型数据库中的表结构。

● 文档 (document)

Elasticsearch中的最小数据单元，常以json格式显示。一个document相当于关系型数据库中的一行数据。

● 倒排索引

一个倒排索引由文档中所有不重复词的列表构成，对于其中每个词，对应一个包含它的文档id列表。

RESTful风格

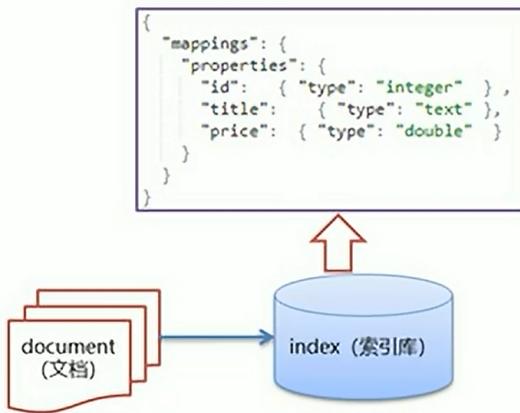
- REST (Representational State Transfer)，表达性状态转移，是一组架构约束条件和原则。满足这些约束条件和原则的应用程序或设计就是RESTful。就是一种定义接口的规范。
- 基于HTTP。
- 可以使用XML格式定义或JSON格式定义。
- 每一个URI代表1种资源。
- 客户端使用GET、POST、PUT、DELETE 4个表示操作方式的动词对服务端资源进行操作：
 - GET：用来获取资源
 - POST：用来新建资源（也可以用于更新资源）
 - PUT：用来更新资源
 - DELETE：用来删除资源

有一个/user资源

- get: /user/1 查询
- delete: /user/1 删除
- post: /user
- update: /user

操作索引

- 添加索引
- 查询索引
- 删除索引
- 关闭索引 ↳
- 打开索引



- 添加索引

```
PUT http://ip:端口/索引名称
```

- 查询索引

```
GET http://ip:端口/索引名称
```

- 删除索引

```
DELETE http://ip:端口/索引名称
```

- 关闭索引

```
POST http://ip:端口/索引名称/_close
```

- 打开索引

```
POST http://ip:端口/索引名称/_open
```

操作映射

简单数据类型

- 字符串
 - text: 会分词, 不支持聚合
 - keyword: 不会分词, 将全部内容作为一个词条, 支持聚合
- 数值
- 布尔
 - boolean
- 二进制
 - binary
- 范围类型
 - integer_range, float_range, long_range, double_range, date_range
- 日期
 - date

操作映射

复杂数据类型

- 数组: []
- 对象: {}

```
{  
  "properties": {  
    "name": { "type": "text" },  
    "age": { "type": "integer" },  
    "address":{  
      "properties":{  
        "province": {"type":"keyword"},  
        "city": {"type":"keyword"},  
        "district": {"type":"keyword"}  
      }  
    }  
  }  
}
```

操作索引

添加映射

```
# 添加映射  
PUT /person/_mapping  
{  
  "properties":{  
    "name":{  
      "type": "text"  
    },  
    "age":{  
      "type": "integer"  
    }  
  }  
}
```

创建索引并添加映射

```
# 创建索引并添加映射  
PUT /person  
{  
  "mappings":{  
    "properties":{  
      "name":{  
        "type": "text"  
      },  
      "age":{  
        "type": "integer"  
      }  
    }  
  }  
}
```

查询映射

```
# 查询映射  
GET /person/_mapping
```

添加字段

```
# 添加字段  
PUT /person/_mapping  
{  
  "properties":{  
    "address":{  
      "type": "text"  
    }  
  }  
}
```

操作文档

添加文档，指定id

```
# 添加文档，指定id  
PUT person/_doc/1  
{  
  "name": "张三",  
  "age": 20,  
  "address": "北京市海淀区"  
}
```

添加文档，不指定id

```
POST person/_doc  
{  
  "name": "张三",  
  "age": 20,  
  "address": "北京市海淀区"  
}
```

查询指定id的文档

```
# 查询指定id的文档  
GET person/_doc/1
```

查询所有文档

```
# 查询所有文档  
GET person/_search
```

删除指定id文档

```
# 删除指定id文档  
DELETE person/_doc/1
```

分词器

- 分词器 (Analyzer) : 将一段文本, 按照一定逻辑, 分析成多个词语的一种工具
如: 华为手机 ---> 华为、手、手机
- ElasticSearch 内置分词器
 - Standard Analyzer - 默认分词器, 按词切分, 小写处理
 - Simple Analyzer - 按照非字母切分(符号被过滤), 小写处理
 - Stop Analyzer - 小写处理, 停用词过滤(the,a,is)
 - Whitespace Analyzer - 按照空格切分, 不转小写
 - Keyword Analyzer - 不分词, 直接将输入当作输出
 - Patter Analyzer - 正则表达式, 默认\W+(非字符分割)
 - Language - 提供了30多种常见语言的分词器
- ElasticSearch 内置分词器对中文很不友好, 处理方式为: 一个字一个词

查询文档

- 词条查询: term
 - 词条查询不会分析查询条件, 只有当词条和查询字符串完全匹配时才匹配搜索
- 全文查询: match
 - 全文查询会分析查询条件, 先将查询条件进行分词, 然后查询, 求并集

```
GET person/_search
{
  "query": {
    "term": {
      "name": {
        "value": "张三"
      }
    }
  }
}

GET person/_search
{
  "query": {
    "match": {
      "address": "北京"
    }
  }
}
```

批量操作-脚本

Bulk 批量操作是将文档的增删改查一些列操作, 通过一次请求全都做完, 减少网络传输次数。

语法:

```
POST /_bulk
{"action": {"metadata"}}
{"data"}
```

示例:

```
POST _bulk
{"delete":{ "_index":"person", "_id":"5" }}
{"create":{ "_index":"person", "_id":"5" }}
{"name":"六号","age":20,"address":"北京"}
{"update":{ "_index":"person", "_id":"2" }}
{"doc":{"name":"二号"}}
```

matchAll查询-脚本

matchAll查询：查询所有文档

语法

```
GET 索引名称/_search
{
  "query": {
    "match_all": {}
  }
}
```

term查询-脚本

term查询：不会对查询条件进行分词。

语法

```
GET 索引名称/_search
{
  "query": {
    "term": {
      "字段名称": {
        "value": "查询条件"
      }
    }
  }
}
```

match查询-脚本

match查询：

- 会对查询条件进行分词。
- 然后将分词后的查询条件和词条进行等值匹配
- 默认取并集 (OR)

语法

```
GET 索引名称/_search
{
  "query": {
    "match": {
      | "字段名称": "查询条件"
    }
  }
}
```

模糊查询-脚本

wildcard查询：会对查询条件进行分词。还可以使用通配符? (任意单个字符) 和 * (0个或多个字符)

regexp查询：正则查询

prefix查询：前缀查询

queryString查询-脚本

queryString:

- 会对查询条件进行分词。
- 然后将分词后的查询条件和词条进行等值匹配
- 默认取并集 (OR)
- 可以指定多个查询字段

```
GET 索引名称/_search
{
  "query": {
    "query_string": {
      "fields": ["字段1","字段2"...],
      "query": "查询条件1 OR 查询条件2"
    }
  }
}
```

```
①
② # 范围查询
③
④ GET goods/_search
⑤ -
⑥   "query": {
⑦     "range": {
⑧       "price": {
⑨         "gte": 2000,
⑩         "lte": 3000
⑪       }
⑫     }
⑬   }
⑭ }
```

布尔查询-脚本

boolQuery: 对多个查询条件连接。连接方式:

- must (and) : 条件必须成立
- must_not (not) : 条件必须不成立
- should (or) : 条件可以成立
- filter: 条件必须成立, 性能比must高。不会计算得分

布尔查询-脚本

boolQuery: 对多个查询条件连接。连接方式:

- must (and) : 条件必须成立
- must_not (not) : 条件必须不成立
- should (or) : 条件可以成立
- filter: 条件必须成立, 性能比must高。不会计算得分

```
GET 索引名称/_search
```

```
{  
  "query": {  
    "bool": {  
      "must": [ {} ],  
      "filter": [ {} ],  
      "must_not": [ {} ],  
      "should": [ {} ]  
    }  
  }  
}
```

聚合查询-脚本

- 指标聚合: 相当于MySQL的聚合函数, max、min、avg、sum等
- 桶聚合: 相当于MySQL的group by 操作。不要对text类型的数据进行分组, 会失败。

高亮查询-脚本

高亮三要素:

- 高亮字段
- 前缀
- 后缀

```
GET goods/_search  
{  
  "query": {  
    "match": {  
      "title": "电视"  
    }  
  },  
  "highlight": {  
    "fields": {  
      "title": {  
        "pre_tags": "<aa>",  
        "post_tags": "</aa>"  
      }  
    }  
  }  
}
```

重建索引

随着业务需求的变更，索引的结构可能发生改变。

ElasticSearch的索引一旦创建，只允许添加字段，不允许改变字段。因为改变字段，需要重建倒排索引，影响内部缓存结构，性能太低。

那么此时，就需要重建一个新的索引，并将原有索引的数据导入到新索引中。



ElasticSearch 集群管理



集群介绍



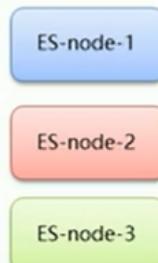
集群和分布式

- 集群：多个人做一样的事。
- 分布式：多个人做不一样的事。

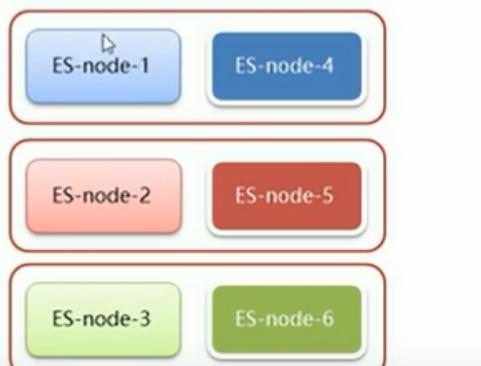
单点架构



集群架构



集群分布式架构



集群解决的问题：

- 让系统高可用
- 分担请求压力

分布式解决的问题：

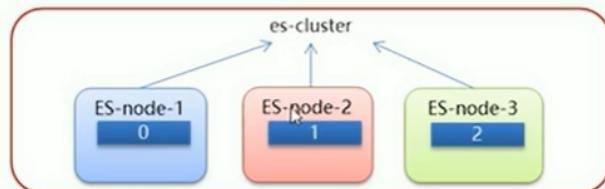
- 分担存储和计算的压力，提速

ElasticSearch 集群特点

- ElasticSearch 天然支持分布式
- ElasticSearch 的设计隐藏了分布式本身的复杂性

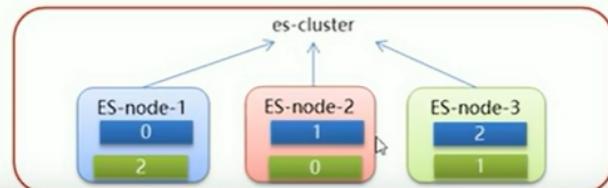
ElasticSearch 集群分布式架构相关概念

- 集群 (cluster) : 一组拥有共同的cluster name 的节点。
- 节点 (node) : 集群中的一个 Elasticsearch 实例
- 索引 (index) : es存储数据的地方。相当于关系数据库中的database概念
- 分片 (shard) : 索引可以被拆分为不同的部分进行存储，称为分片。在集群环境下，一个索引的不同分片可以拆分到不同的节点中



ElasticSearch 集群分布式架构相关概念

- 集群 (cluster) : 一组拥有共同的cluster name 的节点。
- 节点 (node) : 集群中的一个 Elasticsearch 实例
- 索引 (index) : es存储数据的地方。相当于关系数据库中的database概念
- 分片 (shard) : 索引可以被拆分为不同的部分进行存储，称为分片。在集群环境下，一个索引的不同分片可以拆分到不同的节点中
- 主分片 (Primary shard) : 相对于副本分片的定义。
- 副本分片 (Replica shard) 每个主分片可以有一个或者多个副本，数据和主分片一样。



集群原理-分片配置

- 在创建索引时，如果不指定分片配置，则默认主分片1，副本分片1。
- 在创建索引时，可以通过settings设置分片

```
"settings": {  
    "number_of_shards": 3,  
    "number_of_replicas": 1  
},
```

- 分片与自平衡：当节点挂掉后，挂掉的节点分片会自平衡到其他节点中
- 在Elasticsearch中，每个查询在每个分片的单个线程中执行，但是，可以并行处理多个分片。
- 分片数量一旦确定好，不能修改。
- 索引分片推荐配置方案：

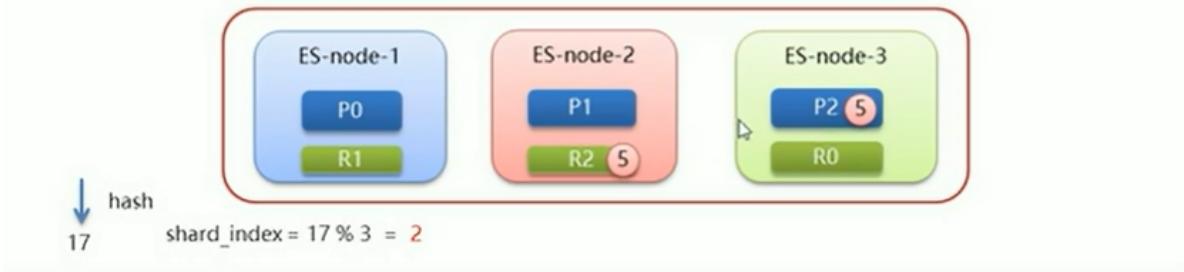
1. 每个分片推荐大小10-30GB
2. 分片数量推荐 = 节点数量 * 1~3倍

思考：比如有1000GB数据，应该有多少个分片？多少个节点

40个分片 20个节点

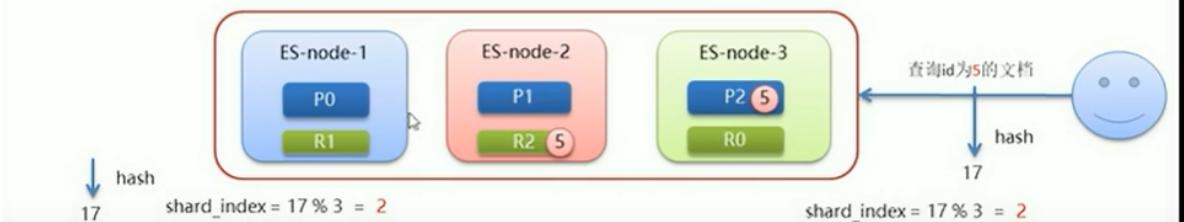
集群原理-路由原理

- 文档存入对应的分片，ES计算分片编号的过程，称为路由。
- Elasticsearch 是怎么知道一个文档应该存放到哪个分片中呢？
- 查询时，根据文档id查询文档， Elasticsearch 又该去哪个分片中查询数据呢？
- 路由算法： $\text{shard_index} = \text{hash(id)} \% \text{number_of_primary_shards}$



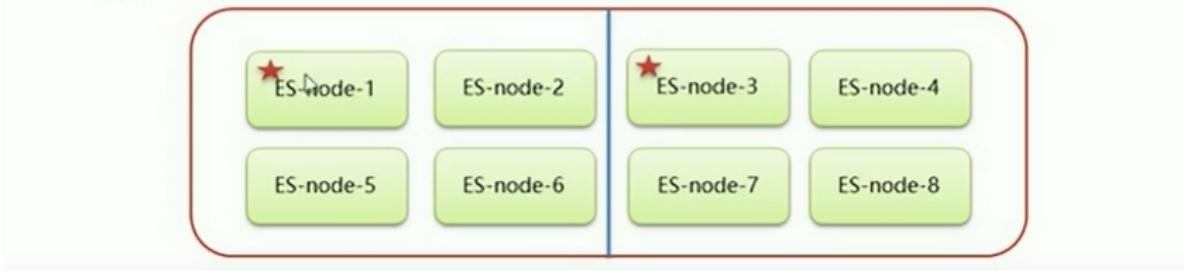
集群原理-路由原理

- 文档存入对应的分片，ES计算分片编号的过程，称为路由。
- Elasticsearch 是怎么知道一个文档应该存放到哪个分片中呢？
- 查询时，根据文档id查询文档， Elasticsearch 又该去哪个分片中查询数据呢？
- 路由算法： $\text{shard_index} = \text{hash(id)} \% \text{number_of_primary_shards}$



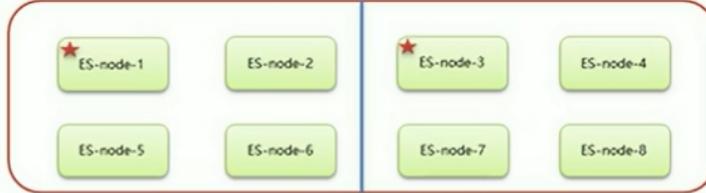
脑裂

- 一个正常es集群中只有一个主节点 (Master)， 主节点负责管理整个集群。如创建或删除索引，跟踪哪些节点是集群的一部分，并决定哪些分片分配给相关的节点。
- 集群的所有节点都会选择同一个节点作为主节点。
- 脑裂问题的出现就是因为从节点在选择主节点上出现分歧导致一个集群出现多个主节点从而使集群分裂，使得集群处于异常状态。



脑裂原因

1. 网络原因：网络延迟
 - 一般es集群会在内网部署，也可能在外网部署，比如阿里云。
 - 内网一般不会出现此问题，外网的网络出现问题的可能性大些。
 2. 节点负载
 - 主节点的角色既为master又为data。数据访问量较大时，可能会导致Master节点停止响应（假死状态）。
- ```
#是不是有资格主节点
node.master: true
#是否存储数据
node.data: true
```
3. JVM内存回收
    - 当Master节点设置的JVM内存较小时，引发JVM的大规模内存回收，造成ES进程失去响应。



## 避免脑裂

脑裂产生的原因：

1. 网络原因：网络延迟较高
2. 节点负载：主节点的角色既为master又为data
3. JVM内存回收：JVM内存设置太小

避免脑裂：

1. 网络原因：discovery.zen.ping.timeout 超时时间配置大一点，默认是35
2. 节点负载：角色分离策略
  - 候选主节点配置为
    - node.master: true
    - node.data: false
  - 数据节点配置为
    - node.master: false
    - node.data: true
3. JVM内存回收：修改 config/jvm.options 文件的 -Xms 和 -Xmx 为服务器的内存一半。